Slide 1

Administrivia

- Homework 5 (I/O) on Web by tomorrow. Due next Thursday.

Slide 2

Minute Essay From Last Lecture

- Anything about I/O that's particularly unclear? that you want to hear more about?
  - When to use SSF and when to use elevator? probably always the latter.
  - Optical disks not important? no, but for this course probably relevant aspects are filesystem, next chapter.
  - I/O in real systems? today.
  - How does it all fit together? soon.

CSCI 4320 November 13, 2003

Slide 3

I/O in Unix/Linux

- Access to devices provided by special files (`/dev/*`), to provide uniform interface for callers. Two categories, block and character. Each defines interface (set of functions) to device driver. Major device number used to locate specific function.
- For block devices, buffer cache contains blocks recently/frequently used. (See figure on p. 729.)
- For character devices, optional line-discipline layer provides some of what we described for text-terminal keyboard driver. (See figure on p. 729.)
- Streams provide additional layer of abstraction for callers — can interface to files, terminals, etc.

Slide 4

I/O in Windows

- Hardware Abstraction Layer (HAL) attempts to insulate rest of o/s from some low-level details — e.g., I/O using ports versus memory-mapped I/O. (See figure p. 779.)
- Standard interface to device drivers — Windows Driver Model. Drivers are passed I/O Request Packet objects. (See figure on p. 829.)
- Interesting comparison of o/s sizes on p. 771.

Slide 5

Files and Filesystems —Overview

- Very abstract view — requirements for long-term information storage are:
  - Store large amounts of information.
  - Have information survive past end of creating process.
  - Allow concurrent access by multiple processes.
- Usual solution — “files” on disk and other external media, organized into “file systems”.
- In terms of the two views of an o/s:
  - “Virtual machine” view — filesystem is important abstraction.
  - “Resource manager” view — filesystem manages disk (and other device) resources.
- We’ll look first at the user view, then at implementation.

Slide 6

File Abstraction

- File names — always “text string”, but some choices: maximum length? case-sensitive? ASCII or Unicode? “extension” required?
- File structure — how file appears to application program:
  - Unstructured sequence of bytes — maximum flexibility, but maybe more work for application.
  - Sequence of fixed-length records — widely used in older systems, not much any more.
  - Tree (or other) structure supporting access by key.

CSCI 4320 November 13, 2003

Slide 7

File Abstraction, Continued

- File types — include “regular files”, also directories and (in some systems, e.g. UNIX) “special files”. Regular files subdivide into:
  - ASCII files — sequences of ASCII characters, generally separated into lines by line-end character(s).
  - Binary files — everything else, including executables (format dictated by o/s’s expectations), various archives, MS Word format, etc., etc.
- File access — sequential versus random-access.
- File attributes — “other stuff” associated with file (owner, protection info, time of creation / last use, etc.)

Slide 8

File Abstraction, Continued

- File operations (things one can do to a file) include create, delete, open, close, read, write, get attributes, set attributes. Example program using system calls on p. 390.

  Also — memory-mapped files (read whole file into memory, process there, write back out).

Slide 9

Directory/Folder Abstraction

- Basic idea — way of grouping / keeping track of files. Can be
  - Single-level (simple but restrictive).
  - Two-level (almost as simple, better if multiple users, but also restrictive).
  - Hierarchical.
- Implies need for path names, which can be absolute or relative (to "working directory").
- Operations on directories include create, delete, open, close, read, add entry, remove entry.

Slide 10

Filesystem Implementation —Overview

- Recall basic organization of disk from chapter 5:
  - Master boot record (includes partition table)
  - Partitions, each containing boot block and lots more blocks.
- How to organize/use those "lots more blocks"? Must keep track of which blocks are used by which files, which blocks are free, directory info, file attributes, etc., etc.

  Typically start with superblock containing basic info about filesystem, then some blocks with info about free space and what files are there, then the actual files.

CSCI 4320 November 13, 2003

Slide 11

Implementing Files —Contiguous Allocation

- Key idea — what the name suggests, much like analogous idea for memory management.
- How does it work? simple, fast (for both sequential and random access), but requires knowing in advance how much space, can lead to fragmentation (wasted space on disk).
- Widely used long ago, abandoned, and now useful again for write-once media.

Slide 12

Implementing Files —Linked-List Allocation

- Key idea — organize each file's blocks as a linked list.
- How does it work? less simple, reasonably fast for sequential access but slow for random access, no fragmentation (in the sense of wasted space), somewhat awkward in using some of disk block for pointers.

Slide 13

Implementing Files —Linked-List Allocation With Table In Memory

- Key idea — keep linked-list scheme, but use table in memory (File Allocation Table or FAT) for pointers rather than using part of disk blocks.
- How does it work? less simple, reasonably fast for both sequential access and random access, no fragmentation (in the sense of wasted space), doesn't require using part of disk block for pointers, may need a lot of memory.

Slide 14

Implementing Files —I-Nodes

- Key idea — associate with each file a data structure ("index node" or i-node) containing file attributes and disk block numbers, keep in memory.
- How does it work? less simple, reasonably fast for both sequential access and random access, no fragmentation (in the sense of wasted space), doesn't require using part of disk block for pointers, doesn't need a lot of memory.

CSCI 4320 November 13, 2003

Slide 15

Implementing Directories

- Main function of directory entry is to allow "open file" system call to map file name to whatever is needed to find disk blocks (first block, i-node, etc.).
- Directory entry should also provide a way to find file attributes. How?
  - Store directly in directory entries (MS-DOS/Windows).
  - Store elsewhere, and provide a pointer from directory entries (Unix).
- Simplest scheme — fixed-size directory entries. Doesn't work well if file names can be long. How to fix?
  - Variable-size entries.
  - Fixed-size entries with pointers into "heap" of file names.

Slide 16

Implementing Shared Files

- What if we want to share files among users? i.e., hierarchy is not a tree?
- One way — directory entries don't point to actual disk blocks, but to data structure containing them (i-node), so can have multiple entries pointing to same file (Unix "hard links").
- Another way — special file ("symbolic link") pointing to actual file.
- Each approach has potential problems . . .

Minute Essay

- One student thought Homework 4 was much harder than previous assignments. Do you agree? If so, why?